A Monte Carlo method based on antithetic variates for network reliability computations

Mohamed El Khadiri, Gerardo Rubino

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1 9 9 2

ème anniversaire

N° 1609

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

A MONTE CARLO METHOD BASED ON ANTITHETIC VARIATES FOR NETWORK RELIABILITY COMPUTATIONS

Mohamed EL KHADIRI
Gerardo RUBINO

Février 1992

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

A Monte Carlo Method Based on Antithetic Variates for Network Reliability Computations

Mohamed El Khadiri and Gerardo Rubino, IRISA-INRIA

Programme I
Publication Interne No. 626, Janvier 1992, 28 pages.

Abstract

The exact evaluation of usual reliability measures of communication networks is seriously limited because of the excessive computational time usually needed to obtain them. In the general case, the computation of almost all the interesting reliability metrics are NP-hard problems. An alternative approach is to estimate them by means of a Monte Carlo simulation. This allows to deal with larger models than those that can be evaluated exactly.

In this paper, we propose an algorithm much more performant in time and in precision that the standard Monte Carlo technique. Moreover, it is particularly efficient in the case of highly reliable systems. It will be shown that it behaves much better than the so called dagger sampling plan on which good results have been reported in the literature. We will also show that the applicability of the dagger method depends on the reliabilities of the components of the network while this is not the case of the method proposed here.

COMUNICATION NETWORKS, NETWORK RELIABILITY, MONTE CARLO SIMULATION, VARIANCE REDUCTION, ANTITHETIC VARIABLES, DAGGER SAMPLING PLANS.

Calcul d'indices de fiabilité de réseaux par une méthode de type Monte Carlo basée sur des variables "antagonistes"

Résumé

Le calcul exact des indices usuels de fiabilité de réseaux est, dans le cas général, un problème NP-difficile. Une alternative est de les évaluer par des simulations de type Monte Carlo. Dans cet article, nous proposons un algorithme beaucoup plus performant en temps et en précision que la méthode de Monte Carlo standard. De plus, il est particulièrement efficace dans le cas des systèmes hautement fiables. Nous montrons que notre algorithme est plus performant que la technique "dagger" sur laquelle de bons résultats ont été publiés. Nous montrons aussi que l'applicabilité de la méthode "dagger" dépend des fiabilités des composantes du réseau, ce qui n'est pas le cas de l'algorithme proposé.

RÉSEAUX DE COMMUNICATION, FIABILITÉ, MÉTHODES DE MONTE CARLO, RÉDUCTION DE LA VARIANCE, VARIABLES ANTITHÉTIQUES, MÉTHODE "DAGGER".

Contents

1 Introduction

In the reliability evaluation of communication networks, a basic and widely used model considers the network as a graph, $\mathcal{G}$, undirected or directed. With the components (nodes and edges or arcs) we associate binary variables with values in $\{0,1\}$. Let us denote by X_c the variable associated with component c, called the *state* of c. Value 1 means that the component works perfectly and value 0 means that it is down and that it can be considered as deleted from the graph. The state of the system is completely characterized by the vector X whose components are the X_c's. In a stochastic context, the X_c's are random variables and it is usually supposed that they are independent. Let us denote by r_c the probability $\Pr(X_c = 1)$, called the *reliability* or *elementary reliability* of c. The states of the set of components of $\mathcal{G}$ define a (random) partial graph of $\mathcal{G}$. A qualitative criteria is then used to evaluate some aspect of the communications in the network. For instance, we can fix two nodes and evaluate the probability that they are connected in the random partial graph (this is called the *2-terminal* reliability of the network.) Another important measure is the probability that every pair of nodes of the graph remains connected, that is, the probability that the resulting partial graph contains all the nodes and that it is connected (this is the *all terminal* reliability index.)

From a general point of view, the qualitative criteria defining what is an operational state of the system, is formalized as a binary function, Φ, called the *structure* function of the system, which associates with each vector X of states of the components, value 1 or value 0 according to the fact that the network is considered to work correctly or to fail in that particular configuration of its components [1]. For instance, in the 2-terminal reliabiliy case, $\Phi(X) = 1$ if and only if the components whose value are 1 in X define a partial graph of $\mathcal{G}$ containing at least the two choosen nodes and one path relying them. Our model is an example of a *monotone* system. Monotone systems are characterized by the following properties: if we denote by $\Phi(0)$ (respectively $\Phi(1)$) the value of Φ when all the components are down (respectively work) and by $(X|X_c = 0)$ (respectively $(X|X_c = 1)$) the vector obtained by replacing the value of X_c in X by 0 (respectively by 1), then $\Phi(0) = 0$, $\Phi(1) = 1$ and for any state X and any component c, $\Phi(X|X_c = 0) \leq \Phi(X|X_c = 1)$. This last relation means that a monotone system can not be degradated by replacing a failed component by a working one. Moreover, we will not only consider the monotonicity property but also assume that the model is *coherent*, which means that for any component c there exists a state X such that $\Phi(X|X_c = 0) < \Phi(X|X_c = 1)$. In other words, all the components are relevant from the point of view of the criteria formalized by Φ. We will denote by R the selected reliability measure, that is, $R = \Pr(\Phi(X) = 1) = \mathrm{E}(\Phi(X))$.

There are several variations in this general framework. For instance, we can distinguish between undirected graphs and directed graphs or digraphs, between the case when both edges (or arcs) and nodes can be down (state 0), or the case in which only one of this two classes of components is subject to failures (the other type of element being considered as perfect.) In another important family of problems, it is assumed that all the components are identical from the reliability point of vue (that is, for all c, $r_c = r$), etc. There are also several different measures of the reliability for the whole system. The exact computation of most of them in the different contexts is a NP–hard problem in the general case [2]. The interested reader can see [3] for related informations and references.

The previous observations explain the difficulties in the evaluation of moderately large systems (say, more than one hundred elements.) An alternative option is to estimate the reliability measures by means of a Monte Carlo technique [4], [5], [6]. In the general case, this works much faster when the network is not very small. The price to pay is to obtain a probabilistic answer, that is, a statistical

estimate of the considered reliability measure. This is the approach that will be investigated here.

In this paper, we adress the following class of models. The graph $\mathcal{G}$ is pressumed to be undirected since this is the case in most models (bidirectional communication channels.) We also assume that the nodes are perfect, only the edges can be in state 0. These are not basic assumptions: The algorithms discussed here can be easily adapted to hold directed graphs with imperfect nodes. In fact, as we shall see, they can be adapted to evaluate the reliability corresponding to any monotone structure function Φ provided that the user can compute it for any vector state. As a consequence, the states of the edges define a (random) subgraph $\widetilde{\mathcal{G}}$ of $\mathcal{G}$ and the structure function Φ is defined on $\{0,1\}^E$ if E denotes the number of edges. The choosen reliability measure R is the following: given a subset $\mathcal{K}$ of the set of nodes, R is the probability that the nodes in $\mathcal{K}$ are connected in $\widetilde{\mathcal{G}}$, that is, the probability that they belong to the same connected component of $\widetilde{\mathcal{G}}$. As particular cases, this measure includes the 2-terminal reliability when $\mathcal{K}$ has two elements, and the all terminal reliability when $\mathcal{K}$ contains all the nodes of the model. To avoid trivialities, let us assume that $0 < r_e < 1$ for all line e. This is not a real restriction: If for some e we have $r_e = 0$, the line can be deleted from the graph and if $r_e = 1$ it can be "contracted": both modifications do not change the value of R.

As previous work on the same problem, we have the method proposed in [7] that we will encounter in Subsection 4.2. In [8], four algorithms for the 2–terminal problem are compared. Two of the four methods require as supplementary input data lists of minpaths or mincuts of the graph. The more efficient of the two other techniques is precisely the one in [7]. Other references on this topic are [9], [6], [10].

The paper is organized as follows. The next section recalls the principles of the Monte Carlo method and the notion of "blocks." Section 3 introduces the use of antithetic variables, a well known technique allowing the construction of estimators with a smaller variance than in "crude" methods [4]. We then present the same idea in a different form allowing a generalization made in Section 4. It is shown that the so called dagger sampling plan proposed in [7] is a particular case. The dagger technique has problems discussed in the same section. The presentation made in our paper allows to solve them in a efficient way. This and further improvements is the content of Section 5 and some numerical results are presented in Section 6. The last section of the paper is devoted to some conclusions.

2 Basics

2.1 The Crude Monte Carlo Technique

The estimation of R by a standard or *crude* Monte Carlo technique consists of generating N independent samples $X^1, \ldots, X^N$ of X and evaluate the unknow parameter R by the unbiased estimator $\widehat{R}$,

$$\widehat{R} = \frac{1}{N} \sum_{n=1}^{N} \Phi(X^n). \tag{1}$$

The variance of this estimator is

$$\mathrm{Var}(\widehat{R}) = \frac{\mathrm{Var}(\Phi(X))}{N} = \frac{R(1-R)}{N}.$$

In order to estimate it we use the unbiased estimator $\widehat{V}$,

$$\widehat{V} = \frac{\widehat{R}(1-\widehat{R})}{N-1}. \tag{2}$$

2.2 Working by "Blocks"

Assume that we generate B independent samples of a block consisting of L variables $X^1,\ldots,X^L$ distributed as X. Let us denote by $X^{(b,1)},\ldots,X^{(b,L)}$ the L variables in block b and associate with them the partial estimator $\widehat{R}_b$,

$$\widehat{R}_b = \frac{1}{L}\sum_{l=1}^{L}\Phi(X^{(b,l)}).$$

This can be seen as generating B independent samples of the random variable M defined by

$$M = \frac{1}{L}\sum_{l=1}^{L}\Phi(X^l).$$

In order to estimate R we use the unbiased estimator $\widehat{R}'$,

$$\widehat{R}' = \frac{1}{B}\sum_{b=1}^{B}\widehat{R}_b. \tag{3}$$

If $X^1,\ldots,X^L$ are independent, we have

$$\mathrm{Var}(M) = \frac{1}{L}\mathrm{Var}(\Phi(X))$$

and consequently

$$\mathrm{Var}(\widehat{R}') = \frac{1}{B}\mathrm{Var}(M) = \frac{1}{LB}\mathrm{Var}(\Phi(X)) = \mathrm{Var}(\widehat{R}).$$

So, there is no gain in proceeding by blocks in this way. Indeed, we have the same variance as a *crude* method for the same sample size $N = BL$. Assume now that the variables $X^1,\ldots,X^L$ are no more independent. So,

$$\mathrm{Var}(M) = \frac{1}{L}\mathrm{Var}(\Phi(X)) + \frac{2}{L^2}\sum_{1\le i<j\le L}\mathrm{Cov}(\Phi(X^i),\Phi(X^j))$$

and

$$\begin{aligned}\mathrm{Var}(\widehat{R}') &= \frac{\mathrm{Var}(\Phi(X))}{BL} + \frac{2}{BL^2}\sum_{1\le i<j\le L}\mathrm{Cov}(\Phi(X^i),\Phi(X^j)) \\ &= \mathrm{Var}(\widehat{R}) + \frac{2}{BL^2}\sum_{1\le i<j\le L}\mathrm{Cov}(\Phi(X^i),\Phi(X^j)).\end{aligned} \tag{4}$$

If for all variables X^i and X^j we have $\mathrm{Cov}(\Phi(X^i),\Phi(X^j)) \le 0$ and if there exists some i and j so that the inequality is strict ($<$), then relation (4) implies that $\mathrm{Var}(\widehat{R}') < \mathrm{Var}(\widehat{R})$. In other words, in such a case $\widehat{R}'$ gives a more accurate estimate than the standard estimator $\widehat{R}$ with the same sample size $N = BL$. The following theorem gives a condition leading to a negative correlation between two variables $\Phi(X^i)$ and $\Phi(X^j)$. This result is the basis of the variance reduction techniques discussed in this paper. The proof is given in the Annex.

Theorem 2.1 *Let Φ be a coherent structure function. If for each e, the variables X_e^i and X_e^j are either negatively correlated or independent and if for at least one edge e we have $Cov(X_e^i, X_e^j) < 0$, then $Cov(\Phi(X^i),\Phi(X^j)) < 0$.*

It remains the problem of estimating the variance of $\mathrm{Var}(\widehat{R}')$. As the standard unbiased estimator of $\mathrm{Var}(M)$ is

$$\frac{1}{B-1}\sum_{b=1}^{B}(\widehat{R}_b - \widehat{R}')^2,$$

we can use the following unbiased estimator for $\mathrm{Var}(\widehat{R}')$:

$$\begin{aligned}\widehat{V}' &= \frac{1}{B(B-1)}\sum_{b=1}^{B}(\widehat{R}_b - \widehat{R}')^2 \\ &= \frac{1}{B(B-1)}\sum_{b=1}^{B}\widehat{R}_b^2 - \frac{1}{B-1}\widehat{R}'^2. \end{aligned} \tag{5}$$

In the following, we will see how this can be obtained and implemented without increasing the computational complexity of the algorithm.

3 Antithetic Variables

Before recalling here the antithetic variables method which is a well known tool in reduction variance techniques, let us be more specific in the presentation of the algorithms. The crude Monte Carlo procedure will be presented as follows.

```
CRUDE MONTE CARLO PROCEDURE

R := 0
FOR n := 1,N DO
   FOR e := 1,E DO
      U := Unif(0,1)
      X[e] := IF U < r[e] THEN 1 ELSE 0
   ENDFOR
   R := R + Φ(X[.])
ENDFOR
R := R/N ; V := R(1 - R)/(N - 1)
```

For each edge e, its elementary reliability r_e is stored in `r[e]`. The function `Unif(0,1)` returns a (pseudo)random number uniformly distributed on the interval [0,1] and the results of the successive calls to this function are pressumed to be independent. We have denoted here by `X[e]` the implementation of the random variable X_e^n for any n, which is the state of edge e in the nth realization. We denote by `X[.]` the implementation of the whole random vector X^n. In the output, `R` has the obtained value of the estimator $\widehat{R}$ (relation (1)) and `V` the obtained value of $\widehat{V}$ (relation (2).)

3.1 The Antithetic Variables Method

If U is a random variable with uniform distribution on [0,1], then the random variable $1-U$ is also uniformly distributed on the same interval. This suggests to use a single call to the function `Unif` to get two states of each edge. This technique is quite general in Monte Carlo theory (see [4] or [5].) Specifically, in the network reliability context the idea can be formalized in the following algorithm in which the sampling is made by blocks of length $L = 2$.

```
MONTE CARLO PROCEDURE USING ANTITHETIC VARIABLES, Version 1

R := 0 ; V := 0
FOR b := 1,B DO
   FOR e := 1,E DO
      U := Unif(0,1)
      X[1][e] := if U < r[e] THEN 1 ELSE 0
      X[2][e] := if 1 - U < r[e] THEN 1 ELSE 0
   ENDFOR
   Rb := (Φ(X[1][.]) + Φ(X[2][.]))/2
   R := R + Rb ; V := V + Rb²
ENDFOR
R := R/B ; V := V/(B(B - 1)) + R²/(B - 1)
```

For each block b we store the two samples $X^{(b,1)}$ et $X^{(b,2)}$ in the array `X[1][.]` and `X[2][.]`. The variable `Rb` corresponds to the estimator $\widehat{R}_b$ defined in Subsection 2.2, `R` has the value of the estimator $\widehat{R}'$ (relation (3)) and `V` the value of $\widehat{V}$ (relation (5).)

The B couples $(X_e^{(b,1)}, X_e^{(b,2)})$, $1 \leq b \leq B$, are independent samples of (X_e^1, X_e^2) which is distributed as follows:

$$\begin{aligned} \Pr(X_e^1 = 1, X_e^2 = 0) &= \Pr(U < r_e, 1 - U \geq r_e) \\ &= \Pr(U < r_e, U \leq 1 - r_e). \end{aligned}$$

If $r_e < 0.5$, we have

$$\Pr(X_e^1 = 1, X_e^2 = 0) = \Pr(U < r_e) = r_e$$

and if $r_e \geq 0.5$,

$$\Pr(X_e^1 = 1, X_e^2 = 0) = \Pr(U \leq 1 - r_e) = 1 - r_e.$$

In the same way we compute the other values of the joint distribution. The result is

$$\begin{aligned} \text{if } r_e < 0.5, \quad \Pr(X_e^1 = 1, X_e^2 = 0) &= \Pr(U \in [0, r_e[) = r_e, \\ \Pr(X_e^1 = 0, X_e^2 = 1) &= \Pr(U \in [1 - r_e, 1[) = r_e, \qquad (6) \\ \Pr(X_e^1 = 0, X_e^2 = 0) &= \Pr(U \in [r_e, 1 - r_e[) = 1 - 2r_e, \qquad (7) \\ \Pr(X_e^1 = 1, X_e^2 = 1) &= 0; \\ \text{if } r_e \geq 0.5, \quad \Pr(X_e^1 = 0, X_e^2 = 1) &= \Pr(U \in [r_e, 1[) = 1 - r_e, \qquad (8) \\ \Pr(X_e^1 = 1, X_e^2 = 0) &= \Pr(U \in [0, 1 - r_e[) = 1 - r_e, \qquad (9) \\ \Pr(X_e^1 = 1, X_e^2 = 1) &= \Pr(U \in [1 - r_e, r_e[) = 2r_e - 1 \qquad (10) \\ \Pr(X_e^1 = 0, X_e^2 = 0) &= 0. \end{aligned}$$

Consequently, we obtain that,

$$\begin{aligned} \mathrm{Cov}(X_e^1, X_e^2) &= \mathrm{E}(X_e^1 X_e^2) - \mathrm{E}(X_e^1)\mathrm{E}(X_e^2) \\ &= \begin{cases} -r_e^2 & \text{if } r_e < 0.5, \\ -(1 - r_e)^2 & \text{if } r_e \geq 0.5. \end{cases} \end{aligned}$$

Since $\mathrm{Cov}(X_e^1, X_e^2) < 0$ for each line e (recall that we assume $0 < r_e < 1$ for all line e), we obtain $\mathrm{Cov}(\Phi(X^1), \Phi(X^2)) < 0$ by using Theorem 2.1. It follows from the relation (4) that this technique produces a small variance than crude method for the same sample size. Moreover, the random number generator is used Nx$E/2$ times, half of the NxE times needed by the crude method.

3.2 A Different Presentation

Consider the case of $r_e < 0.5$. Since U is uniform, we can rewrite relations (6) et (7) respectively as

$$\Pr(X_e^1 = 0, X_e^2 = 1) = \Pr(U \in [r_e, 2r_e[) = r_e$$

and

$$\Pr(X_e^1 = 0, X_e^2 = 0) = \Pr(U \in [2r_e, 1[) = 1 - 2r_e.$$

In the case of $r_e \geq 0.5$, we replace relations (8), (9) et (10) respectively by

$$\begin{aligned}
\Pr(X_e^1 = 0, X_e^2 = 1) &= \Pr(U \in [0, 1 - r_e[) = 1 - r_e, \\
\Pr(X_e^1 = 1, X_e^2 = 0) &= \Pr(U \in [1 - r_e, 2(1 - r_e)[) = 1 - r_e, \\
\Pr(X_e^1 = 1, X_e^2 = 1) &= \Pr(U \in [2(1 - r_e), 1[) = 2r_e - 1.
\end{aligned}$$

Let us define

$$X_e^{default} = \begin{cases} 0 & \text{if } r_e < 0.5 \\ 1 & \text{if } r_e \geq 0.5 \end{cases} \quad (X_e^{default} \text{ is the most probable state of edge } e),$$

$$q_e = \begin{cases} r_e & \text{if } r_e < 0.5 \\ 1 - r_e & \text{if } r_e \geq 0.5 \end{cases} \quad \text{(observe that } q_e \leq 0.5),$$

$I_e^1 = [0, q_e[$, $I_e^2 = [q_e, 2q_e[$, $J_e^2 = [2q_e, 1[$.

The random variable X_e^i, $i = 1, 2$, can then be written

$$X_e^i = (1 - X_e^{default}) 1_{(U \in I_e^i)} + X_e^{default} 1_{(U \notin I_e^i)}$$

where 1_A denotes the indicator function of the boolean A. It follows the second version of the Monte Carlo algorithm using the antithetic variables method:

```
MONTE CARLO PROCEDURE USING ANTITHETIC VARIABLES, Version 2

R := 0 ; V := 0
FOR b := 1,B DO
   FOR e := 1,E DO
      X[1][e] := X[2][e] := Xdefault[e]  --- initializations
      U := Unif(0,1)
      CASE
         U ∈ I_e^1 : X[1][e] := 1 - Xdefault[e]
         U ∈ I_e^2 : X[2][e] := 1 - Xdefault[e]
         DEFAULT :  --- here, U ∈ J_e^2 ; the initial values remain
      ENDCASE
   ENDFOR
   Rb := (Φ(X[1][.]) + Φ(X[2][.]))/2
   R := R + Rb ; V := V + Rb^2
ENDFOR
R := R/B ; V := V/(B(B - 1)) + R^2/(B - 1)
```

Observe that since the distribution of the vector (X_e^1, X_e^2) has not been changed, the statistical properties are the same for the two versions. This algorithm is generalized in the following section and improved in Section 5.

4 Generalizing the Antithetic Variables Method

4.1 Sampling Edges States

Suppose that the length of the last interval J_e^2 is greater than q_e. We can split it into the two subintervals $[2q_e, 3q_e[$, denoted by I_e^3, and $[3q_e, 1[$, denoted by J_e^3. Let us define the three random variables X_e^i, $i = 1, 2, 3$, by means of

$$X_e^i = (1 - X_e^{default}) 1_{(U \in I_e^i)} + X_e^{default} 1_{(U \notin I_e^i)}. \tag{11}$$

Since the length of I_e^i is equal to r_e, each X_e^i is distributed as X_e. If the length of J_e^3 is greater than q_e, we can construct a fourth interval I_e^4 and the corresponding variable X_e^4. This procedure can be made at most $\lfloor 1/q_e \rfloor$ times where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to the real number x. So, we can choose any s_e verifying $1 \leq s_e \leq \lfloor 1/q_e \rfloor$ and generate s_e states for X_e with a single uniform random number using

$$X_e^i = (1 - X_e^{default}) 1_{(U \in I_e^i)} + X_e^{default} 1_{(U \notin I_e^i)} \quad i = 1, \ldots, s_e. \tag{12}$$

In the following, we show that for every choice $s_e \geq 2$ and any $1 \leq i < j \leq s_e$, we have $\mathrm{Cov}(X_e^i, X_e^j) = -q_e^2 < 0$.

From (12), we verify that

$$\mathrm{E}(X_e^i) = \begin{cases} 1 - q_e & \text{if } X_e^{default} = 1 \\ q_e & \text{if } X_e^{default} = 0 \end{cases} = r_e.$$

For any (i, j) such that $1 \leq i < j \leq s_e$ the couple (X_e^i, X_e^j) is distributed as follows:

$$\begin{aligned} \Pr(X_e^i = 1 - X_e^{default}, X_e^j = X_e^{default}) &= \Pr(U \in I_e^h) = q_e, \\ \Pr(X_e^i = X_e^{default}, X_e^j = 1 - X_e^{default}) &= \Pr(U \in I_e^k) = q_e, \\ \Pr(X_e^i = X_e^{default}, X_e^j = X_e^{default}) &= \Pr(U \notin I_e^h \cup I_e^k) = 1 - 2q_e, \\ \Pr(X_e^i = 1 - X_e^{default}, X_e^j = 1 - X_e^{default}) &= 0 \end{aligned}$$

and the value of $\mathrm{Cov}(X_e^i, X_e^j)$ follows:

$$\mathrm{Cov}(X_e^i, X_e^j) = -q_e^2 < 0. \tag{13}$$

In Section 6 the discussion takes into account the coefficient of correlation between X_e^i and X_e^j which is obtained by dividing (13) by $r_e(1 - r_e) = q_e(1 - q_e)$:

$$\rho(X_e^i, X_e^j) = -\frac{q_e}{1 - q_e}. \tag{14}$$

4.2 Sampling System States: the Dagger Algorithm

For each line e, the choice of s_e must verify $1 \leq s_e \leq \lfloor 1/q_e \rfloor$. Since $q_e \leq 0.5$, we have $\lfloor 1/q_e \rfloor \geq 2$. This implies that the choices $s_e = 1$ for all line e and $s_e = 2$ for all line e are always possible. They give respectively the crude Monte Carlo method and the Monte Carlo method using antithetic variables presented in Section 3.

The maximal choice for each s_e gives the method proposed in [7]. In that paper, there are few details about the block construction procedure; in the only proposed example all the elementary

reliabilities are identical. This leads to $s_e = s$ for all e and in a straightforward manner, the size of the blocks is equal to s. Moreover, the problem of the estimation of the variance of the proposed estimator is not developped.

In [8], the author takes back this technique and proposes as the block size L, the least common multiple of the $\lfloor 1/q_e \rfloor$'s. Let us call "dagger value" this number and denote it by L_d. For each line e, the L states are decomposed in $nsb_e = L/\lfloor 1/q_e \rfloor$ subblocks of size $s_e = \lfloor 1/q_e \rfloor$. For each subblock, a single random number uniformly distributed on $[0,1]$ allows to assign s_e states of line e by means of (12). Since the L realizations are correlated, B independent blocks are constructed and the estimators $\widehat{R}'$ and $\widehat{V}'$ (see Section 2.2) are used to estimate R and $\mathrm{Var}(\widehat{R}')$.

Let us present more formally the resulting technique which will be called in the sequel, the dagger algorithm, as presented in [8]. In a preprocessing task performed by the `Preliminaries()` procedure, the following parameters are computed for each edge e:

q_e: the probability that the line e is in its less probable state,

s_e: the subblock size, i.e. $s_e = \lfloor 1/q_e \rfloor$,

L: the block size, i.e. the least common multiple of the s_e's,

nbs_e: the number of subblocks per block, $nbs_e = L/s_e$,

$X_e^{default}$: the most probable state of line e, i.e. 1 if $r_e \leq 0.5$, 0 otherwise.

They are stored respectively in the variables `q[e]`, `s[e]`, `L`, `nbs[e]` and `Xdefault[e]`. We use the algorithmic array `X[][]` to store a block of L vectors samples (the component `X[i][e]` stores the state of line e in the ith realization.)

According to (12), at most one sample X_e^i among s_e (that is, in each subblock) will be different from $X_e^{default}$. The first step (`BlockInit()` procedure) consists of initializing of the variables `X[i][e]`'s with `Xdefault[e]`.

```
BlockInit()

FOR e := 1,E DO
   FOR i := 1,L DO X[i][e] := Xdefault[e] ENDFOR
ENDFOR
```

Since the array `X[][]` has ExL components, the time complexity of this step is $O(ExL)$. The second step (`ModifiedStates()` procedure) determines for each line e and each subblock m, $1 \leq m \leq nsb_e$, which state in the subblock (if any) will change its initial value.

```
ModifiedStates()

FOR e := 1,E DO
   FOR m := 1,nsb[e] DO
      U := Unif(0,1)
      IF U < nsb[e]q[e] THEN        --- U ∉ J_e^{s_e}
         h := floor(U/q[e]) + 1     --- U ∈ I_e^h ⟺ h = ⌊U/q_e⌋ + 1
         i := (m - 1)s[e] + h
         X[i][e] := 1 - Xdefault[e]
```

```
      ENDIF
   ENDFOR
ENDFOR
```

The `ModifiedStates()` procedure has a time complexity $O(\sum_{e=1}^{E} nbs_e)$. The main algorithm constructs B blocks (B is here an input parameter) and performs the estimations of R and $\mathrm{Var}(\widehat{R}')$. The obtained values are respectively stored in the variables `R` et `V`. Φ performs a Depth First Search (time complexity $O(E)$) to return 1 iff the nodes in $\mathcal{K}$ are connected.

```
DAGGER SAMPLING PROCEDURE

Preliminaries()
R := 0 ; V := 0   --- initializations
FOR b := 1,B DO
   BlockInit()
   ModifiedStates()
   Rb := (1/L)∑_{l=1}^{L} Φ(X[l][.])
   R := R + Rb ; V := V + Rb²
ENDFOR
R := R/B ; V := V/(B(B - 1)) + R²/(B - 1)
```

4.3 Complexity Aspects of the Dagger Algorithm

The time complexity of the dagger algorithm is

$$O(B\text{x}L\text{x}E) + O(B\sum_{e=1}^{E} nbs_e) + O(B\text{x}L\text{x}E)$$

where we do not consider the effect of the preprocessing task performed by the `Preliminaries()` procedure (whose time complexity is $O(E)$.) It corresponds to B calls to `BlockInit()` and to `ModifiedStates()`, and BxL calls to Φ. Since $\sum_{e=1}^{E} nbs_e < E$xL, we obtain that this complexity is $O(B$xLx$E)$ and that the time complexity per realization is $O(E)$. Observe that in the best case, we have $\sum_{e=1}^{E} nbs_e = E$ (when the reliability r_e does not depend on e) but this does not change the total time complexity per realization due to the other terms.

The dagger algorithm uses the array `X[][]` with ExL components to sample the L vector states within each block, so, it has a spatial complexity $O(E$x$L)$.

4.4 Drawbacks of the Dagger algorithm

The dagger method has a major problem, namely the fact that L can be very large. First at all, it is no difficult to construct examples where L can not be stored in a memory word, in particular in the case of highly reliable components. Even if L can be stored, the spatial complexity of $O(E$x$L)$ is a serious constraint when L is large. Since the approach developped in Subsection 4.2 allows any choice for the s_e's between 1 and $\lfloor 1/q_e \rfloor$, to solve this problem we can for instance assign the size $s_e = 2^{k_e}$ to each line e where $k_e = \min(K, \lfloor \log_2(1/q_e) \rfloor)$ and K is a given integer less than the word size in the used computer ($K < 32$ on a 32-bits workstation.) This choice leads to

$$\begin{aligned} L &= \text{least common multiple of the } s_e\text{'s} \\ &= 2^{\max\{k_e,\ 1 \le e \le E\}} \le 2^K. \end{aligned}$$

Morevoer, another drawback is the following. A high value of L can occur in the case when $\min(R, 1-R)$ is not very small. In such a case, the crude Monte Carlo method allows to obtain accurate estimates of R and of the variance of its estimator with a "reasonable" sample size N. For this total sample size, the number B of blocks in the dagger sampling plan may be not large enough to obtain an accurate estimate of $\mathrm{Var}(\widehat{R}')$. Observe that we can even have L greater than N so that (5) can not be applied (we need at least $B = 2$ to apply it.) We will come back to this problem of high values of L in Section 6.

The approach introduced in the previous subsection gives full freedom in the construction of the blocks and it is the basis for the method presented in the following section.

5 An Efficient Monte Carlo Algorithm

5.1 Principles

We propose to choose any fixed block size L, that is, L is an input in the algorithm. Note that if the choice is $L = 1$ we obtain the crude Monte Carlo method and the case of $L = 2$ corresponds to the classic Monte Carlo method using antithetic variables given in Section 3. For each line e, we compute $s_e = \min(L, \lfloor 1/q_e \rfloor)$. In each block, there will be nsb_e subblocks corresponding to e with $nbs_e = \lceil L/s_e \rceil$. The first $nbs_e - 1$ subblocks will have size s_e and the remaining one will have size $t_e = L - s_e(nsb_e - 1)$. This means that if $L \bmod s_e = 0$ then $t_e = s_e$, otherwise, $t_e = L \bmod s_e$. See that if $L \leq \lfloor 1/q_e \rfloor$ there will be only one subblock corresponding to edge e with size L, that is, $nsb_e = 1$ and $t_e = L$.

The L states of line e are constructed from $nsb_e - 1$ independent samples of the vector defined in (12) and a last sample of (X_e^i) with $1 \leq i \leq t_e$, constructed independently of the previous samples by replacing s_e in (12) by t_e.

Consider now the variables $\Phi(X^i)$ and $\Phi(X^j)$ in the same block, $1 \leq i, j \leq L$, $i \neq j$. If for all line e the variables X_e^i and X_e^j are not in the same subblock, they are independent and $\mathrm{Cov}(\Phi(X^i), \Phi(X^j)) = 0$. Otherwise, there is at least one edge e and two realizations i and j such that X_e^i and X_e^j belong to the same subblock and consequently $\mathrm{Cov}(X_e^i, X_e^j) < 0$ which implies that $\mathrm{Cov}(\Phi(X^i), \Phi(X^j)) < 0$ (see the Annex.) Observe that if the block size L is greater than 1, we have at least $\mathrm{Cov}(\Phi(X^1), \Phi(X^2)) < 0$. As we stated in Subsection 2.2, it results a reduction in the variance of the estimator with respect to the variance of the crude method.

5.2 Optimizing the Memory Requirements

The dagger algorithm given in the previous section has a spatial complexity of $O(E \mathrm{x} L)$ since for each block b, it generates the L state vectors before the use of the Φ function to evaluate them. We propose to use a compact storage of the table `X[] []` by eliminating the second dimension L, leading to a spatial complexity of $O(E)$. Moreover, this will avoid the $B \mathrm{x} L \mathrm{x} E$ initializations performed by the procedure `BlockInit()`. As we will immediately see, this together with the second improvement, will also reduce the computing time.

5.3 Improving the Number of Calls to Φ

The probability that the state vector of the network is equal to $X^{default}$ is

$$\Pr(X = X^{default}) = \prod_{e=1}^{E} (1 - q_e). \tag{15}$$

If this probability is not small, that is, if the elementary reliabilities are high (or small) and the number of links is not too large, we will probably encounter, during the sampling process, state vectors identical to the initial vector $X^{default}$. This suggests to evaluate $Y^{default} = \Phi(X^{default})$ as a preliminary task before the construction of the blocks and to use this value for each state vector X such that $X_e = X_e^{default}$ for all link e. To illustrate the gain of the proposed improvement, let us consider for simplicity the case of all the edges having the same reliability $r \geq 0.5$. This assumption implies that for all edge e, we have $X_e^{default} = 1$ and $q_e = q = 1 - r$. Let us also assign to the block size parameter the value $L = \lfloor 1/q \rfloor$. Let us denote by $C(X)$ the event "a call to function Φ is needed when the vector state is X." We have

$$\Pr(C(X)) = 1 - \Pr\left(X = X^{default}\right).$$

Choosing for instance a network with $E = 26$ components (see Figure 1 in Section 6) with a common reliability $r = 0.9999$, we obtain

$$\Pr(C(X)) = 1 - 0.9999^{26} \approx 2.6 \times 10^{-3}.$$

Another way to diminish the number of calls to the Φ function is to use the fact that the system cannot be down if the number of failed links is less than the size of a minimal mincut of Φ. This number, say c, can be computed in polynomial time (see [11, Section 5.2] for details.) Let us denote by $nbF(X)$ the number of failed edges, that is, the number of edges in state 0 when the vector state is X. We have now

$$\begin{aligned}
\Pr(C(X)) &= \Pr\left(X \neq X^{default} \text{ and } nbF(X) \geq c\right) \\
&= \Pr\left(nbF(X) \geq c\right) \\
&= 1 - \Pr\left(nbF(X) < c\right) \\
&= 1 - \sum_{i=0}^{c-1} \frac{E!}{i!(E-i)!} r^{E-i}(1-r)^i.
\end{aligned}$$

Considering the same example as before, we have $c = 2$ and

$$\Pr(C(X)) = 1 - 0.9999^{26} - 26 \times 0.9999^{25} \times 0.0001 \approx 3.2 \times 10^{-6}.$$

Observe that if c is not available, we can use a lower bound as well. For instance, if the graph is biconnected [12] we have $c \geq 2$ (Menger's Theorem.) Of course, this is only an aspect of the problem since the corresponding overheads must be taken into account. See Section 6 for some numerical results illustrating the effects of the proposed improvements and the gain with respect to the dagger algorithm.

5.4 The General "Antithetic" Algorithm

A `Preliminaries()` procedure receives as input parameters the elementary reliabilities r_e's and the choosen block size L. It derives the q_e's, $X_e^{default}$'s, the s_e's, the nbs_e's and the t_e's parameters. These values are stored respectively in the arrays `q[.]`, `Xdefault[.]`, `s[.]`, `nbs[.]` and `t[.]`. Also, the most probably state of the network $\Phi(X^{default})$ is computed and stored in the variable `Ydefault`. If the size of a minimal mincut of Φ is available (or a lower bound), it is stored in the variable `c`. The `Preliminaries()` procedure computes also the number of failed components in the most probably vector state $X^{default}$, that is, the number $E - \sum_{e=1}^{E} X_e^{default}$ which is stored in the `nbFailedDefault` variable.

The compact storage is done by means of two auxiliary vectors `first[]` and `done[]` of size E and some supplementary scalar variables. Instead of constructing the bi-dimensional table `X[][]`, the idea is to see in a "virtual" table `X[][]` where is the first line or realization where some component e has a value different from $X_e^{default}$. This realization number will be stored in the `next` variable. For each e, the `first[e]` variable will contain the first realization. where $X_e \neq X_e^{default}$ so that `next = min(first[.])`. The `done[e]` variable stores the number of subblocks corresponding to edge e for which the algorithm has already assigned values.

Assume that we are at the beginning of the processing of the first block. We proceed by executing the `InitSampling()` procedure:

```
InitSampling()

FOR e := 1,E DO
   done[e] := 0
   Assign(e)
ENDFOR
```

The `Assign(e)` procedure constructs the next subblocks corresponding to edge e. The construction process stops when there is a subblock having (exactly) one value equal to $1 - X_e^{default}$ or when there is no more subblocks to construct. It also computes the `first[e]` value. The information that for edge e there is no more subblocks having one value different from $X_e^{default}$ is stored by setting `first[e] := L + 1`.

```
Assign(e)

first[e] := L + 1  --- default value
WHILE done[e] < nsb[e] DO  --- at least, it remains one unassigned subblock
         --- only the last subblock has size equal to t[e]
   aux := IF done[e] = nsb[e] - 1 THEN t[e] ELSE s[e]
   SampleNextSubblock(aux,e)
   done[e] := done[e] + 1
   IF first[e] ≠ L + 1 THEN BREAK
ENDWHILE
```

The `BREAK` instruction is executed when a new realization number has been calculated and stored in `first[e]`. The `SampleNextSubblock()` procedure performs the corresponding calls to the pseudo-random numbers generator via the `Unif()` function and computes the value of `first[e]`.

```
SampleNextSubblock(bs,e)

U := unif(0,1)
IF U ∉ [bsxq[e],1]  THEN
   --- U  ∈  I_e^h
   h := floor(U/q[e]) + 1
   first[e] := done[e]s[e] + h
ENDIF
```

In the `SampleNextSubblock()` procedure, the case of `U` $\in$ `[bsxq[e],1]` means that in the current subblock, all states of edge e are equal to $X_e^{default}$. We must now look at the main procedure to understand the whole work.

```
GENERAL ANTITHETIC ALGORITHM

Preliminaries()

R := 0 ; V := 0
FOR b := 1 TO B DO

   --- determine for each edge e the next realization where
   --- the state becomes ≠ Xdefault[e]
   InitSample()
   --- determine the next realization with X ≠ Xdefault
   next := min(first[.])
   previous := 0
   Rb := 0
   WHILE next ≤ L DO
      jump := next - previous - 1
      --- there are 'jump' network states with X[.] = Xdefault[.]
      Rb := Rb + jump*Ydefault
      --- the procedure State() produces in variables X[.] and nbFailed
      --- the state vector system and the number of failed components
      --- at realization 'next'; also, it updaptes the vector first[.]
      State()
      --- we call Φ only if the number of failed components is
      --- greater than c
      Rb := IF nbFailed < c THEN Rb + 1 ELSE Rb + Φ(X[.])
      previous := next
      --- we compute the next replication number verifying X ≠ Xdefault
      next := min(first[.])
   ENDWHILE
   jump := next - previous - 1
   Rb := (Rb + jump*Ydefault)/L
   R := R + Rb ; V := V + Rb^2

ENDFOR
R := R/B
V := V/(B(B - 1)) + R^2/(B - 1)
```

It remains to see the details about the **State()** procedure which determines the state system vector at realization number **'next'** and the number of failed components in the generated vector. The procedure also updates the vector **first[.]** by calling **Assign(e)** for each edge e such that **first[e]** = **next**.

```
State()

--- initialize the number of failed components
nbFailed := nbFailedDefault
```

```
FOR e := 1,E DO
   --- determine state of edge e at realization next
   IF first[e] = next THEN
      X[e] := 1 - Xdefault[e]
      --- update the number of failed components
      nbFailed := IF X[e] = 1 THEN nbFailed - 1 ELSE nbFailed + 1
      --- if it remains at least one subblock to construct,
      --- it is constructed by calling the Assign() procedure who updabtes
      --- the value of first[e].
      IF done[e] < nsb[e] THEN Assign(e) ENDIF
   ELSE
      X[e] := Xdefault[e]
   ENDIF
ENDFOR
```

5.5 Complexity Aspects

To simplify the analysis of the time complexity, we consider the case of all the edges having the same reliability r. As stated before, we can choose any block size L verifying $1 \leq L \leq L_d = \lfloor 1/q \rfloor$. For any fixed choice L, we obtain for each edge only one subblock.

In order to obtain the L state vectors, the **Assign()** procedure will be called exactly once for each edge and it will call exactly once the procedure **SampleNextSubblock()**. This means that the time complexity of the block construction is $O(E)$.

Let us denote

$$p = \Pr(X \neq X^{default}) = 1 - q^E$$

and

$$p_c = \Pr(X \neq X^{default} \text{ and } nbF(X) \geq c).$$

In each block, the procedures **min()** and **state()** will be called on the average pL times. The function Φ, implemented as a Depth First Search, will be called, on the average, $p_c L$ times. Since the three procedures have a time complexity $O(E)$ in the worst case, the mean time complexity per block is

$$O(E) + O(E \times L \times p) + O(E \times L \times p_c).$$

Since $p_c \leq p$, we can write that this time complexity is $O(E) + O(E \times L \times p)$.

The mean time complexity of the algorithm per realization is then

$$O(\alpha E)$$

where

$$\alpha = \max\left(\frac{1}{L}, p\right).$$

This is to be compared to the time complexity per realization of the dagger algorithm as presented in [8], discussed in Subsection 4.3, which is $O(E)$. Observe that this last complexity is $O(E)$ for any value of r while in our algorithm the parameter α depends on r and L.

The spatial complexity of the general antithetic algorithm is simply $O(E)$ since no 2-dimensional table is used.

6 Some Numerical Results

To illustrate the effects of the proposed improvements, let us assume that all the edges have the same reliability $r \geq 0.5$ so that the dagger value for the block size L is simply $L_d = \lfloor 1/(1-r) \rfloor$. We use a version of the well known Arpanet computer network whose topology is shown in Figure 1 and we consider the *source-terminal* reliability measure. Table 1 shows the mean CPU (Sun 4) execution time in milliseconds for one realization, of the dagger algorithm as presented in Subsection 4.2 and of the general antithetic algorithm of Subsection 5.4 with the choice $L = L_d$.

common reliability of edges	mean time (ms) per repl dagger algorithm [8]	mean time (ms) per repl antithetic algorithm	ratio
0.9999999	*	3.70×10^{-7}	
0.999999	*	3.67×10^{-6}	
0.99999	3.46×10^{-1}	3.70×10^{-5}	$9.35\times10^{+3}$
0.9999	3.31×10^{-1}	3.70×10^{-4}	$8.95\times10^{+2}$
0.999	3.27×10^{-1}	3.77×10^{-3}	$8.60\times10^{+1}$
0.99	3.33×10^{-1}	3.97×10^{-2}	8.32
0.9	3.58×10^{-1}	3.63×10^{-1}	9.86×10^{-1}
0.5	6.25×10^{-1}	7.31×10^{-1}	8.55×10^{-1}

Table 1: Influence of the proposed improvements on the CPU execution time of the dagger algorithm [8] and the general antithetic algorithm of Subsection 5.4, for different values of the common elementary reliability of the edges.

Table 1 shows the efficiency of the general antithetic algorithm in the case of highly reliable systems. This efficiency diminishes with the common reliability of the edges, due to the fact that the parameters p and p_c defined in the previous section increase with r. When the value of r is particularly low (0.9 or 0.5 in the reported experiment), there is no gain. This is due to the fact that the improvements have no effect and therefore the overhead introduced by them are not compensated. In the first two cases, marked by '*' in the table, the high value of L leads to a memory problem preventing from executing the dagger algorithm.

Let us consider now the problem of choosing the value of L. Assume that we want to analyze the effects of a poor reliability in just a small area of the Arpanet network given in Figure 1. More specifically, assume that we assign to the elementary reliability of edges e_1 and e_2 the value 0.3 and that a third edge, e_3, has its reliability equal to $1 - 1/2^{18}$ (≈ 0.99999619). The remaining elementary reliabilities are all equal to $1 - 1/251$ (≈ 0.99601594).

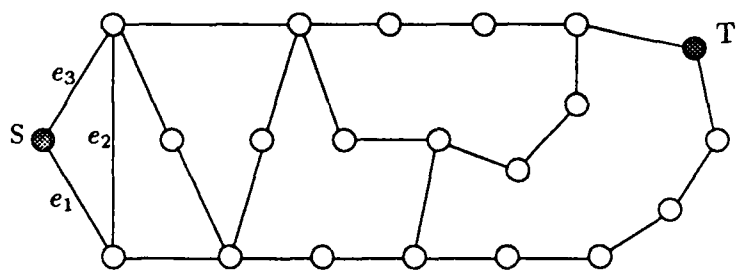

Figure 1: A version of the Arpanet network.

The exact value of R (obtained from the factoring algorithm proposed in [13]) is $R = 0.99989907$.

This exact value allows to validate the estimate of R. Also, in order to underline the variance reduction point of view for N realizations we use the exact value of the variance of the "crude" estimator $\widehat{R}$, that is, the number $R(1-R)/N$.

Our approach allows us to assign any value to L. We give in Tables 2 and 3 some results corresponding respectively to the choices $L = 3$ and $L = 3\text{x}251 = 753$.

number B of blocks	number N of realizations	estimate of variance	$R(1-R)/N$	5% confidence interval	total time in seconds
$1\text{x}10^{+6}$	$30\text{x}10^{+5}$	$2.96\text{x}10^{-11}$	$3.36\text{x}10^{-11}$	]0.99990068, 0.99992199[	$1.06\text{x}10^{+3}$
$1.5\text{x}10^{+6}$	$45\text{x}10^{+5}$	$2.01\text{x}10^{-11}$	$2.24\text{x}10^{-11}$	]0.99990054, 0.99991813[	$1.59\text{x}10^{+3}$
$2\text{x}10^{+6}$	$60\text{x}10^{+5}$	$1.54\text{x}10^{-11}$	$1.68\text{x}10^{-11}$	]0.99989963, 0.99991503[	$2.12\text{x}10^{+3}$
$2.5\text{x}10^{+6}$	$75\text{x}10^{+5}$	$1.25\text{x}10^{-11}$	$1.35\text{x}10^{-11}$	]0.99989906, 0.99991294[	$2.66\text{x}10^{+3}$

Table 2: Results corresponding to the choice $L = 3$ for different total number of blocks

number B of blocks	number N of realizations	estimate of variance	$R(1-R)/N$	5% confidence interval	total time in seconds
$1.2\text{x}10^{+4}$	$90.36\text{x}10^{+5}$	$1.02\text{x}10^{-11}$	$1.12\text{x}10^{-11}$	]0.99990243, 0.99991496[	$1.00\text{x}10^{+3}$
$1.8\text{x}10^{+4}$	$135.54\text{x}10^{+5}$	$6.91\text{x}10^{-12}$	$7.45\text{x}10^{-12}$	]0.99990167, 0.99991197[	$1.51\text{x}10^{+3}$
$2.4\text{x}10^{+4}$	$180.72\text{x}10^{+5}$	$5.33\text{x}10^{-12}$	$5.58\text{x}10^{-12}$	]0.99989831, 0.99990736[	$2.02\text{x}10^{+3}$
$3\text{x}10^{+4}$	$225.90\text{x}10^{+5}$	$4.30\text{x}10^{-12}$	$4.47\text{x}10^{-12}$	]0.99989766, 0.99990579[	$2.52\text{x}10^{+3}$

Table 3: Results corresponding to the choice $L = 753$ for different total number of blocks

These results show that the variance reduction corresponding to each of the two values of L is very small. In Figures 2 and 3 we plot the ratio $\widehat{V}'/(R(1-R)/N)$ in % against N for the two values $L = 3$ and $L = 753$. We can see that the best ratio is about 90 %.

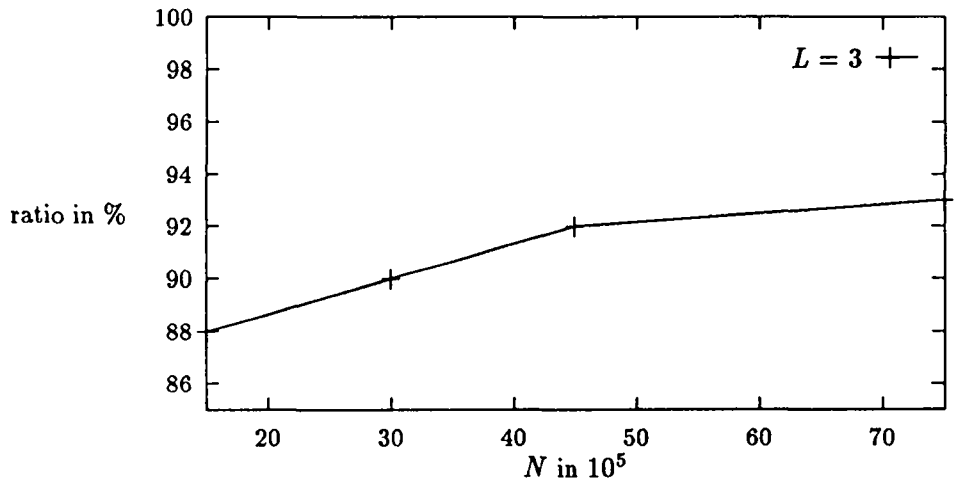

Figure 2: Ratio $\dfrac{\widehat{V}'}{R(1-R)/N}$ against the number N of realizations, for $L = 3$.

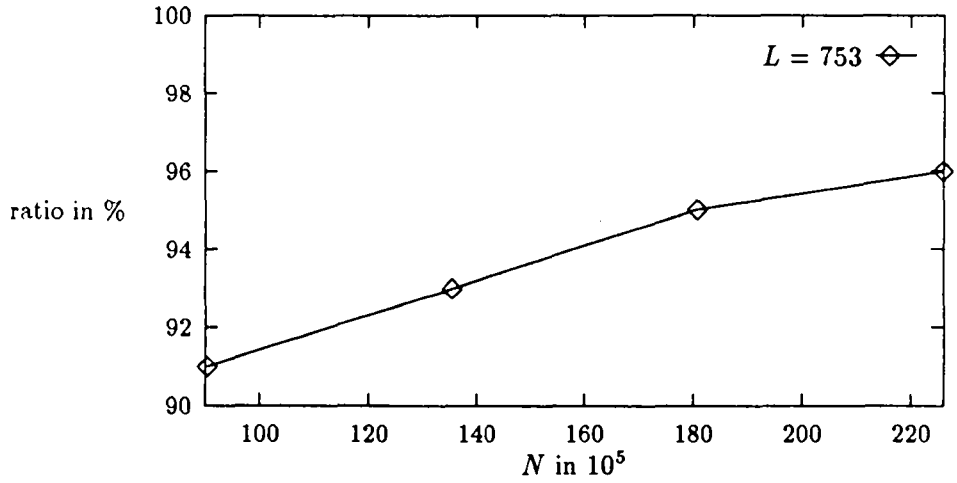

Figure 3: Ratio $\frac{\widehat{V}'}{R(1-R)/N}$ against the number N of realizations, for $L = 753$.

In Figure 4 we plot the curves variance–time associated with the values $L = 3$ and $L = 753$. The information given by this figure is that the performance of the value $L = 753$ is better than the value $L = 3$. From Figures 2 and 3 we can conclude that this better performance is only due to the execution time reduction.

When L is larger than 753, the variance reduction can be only improved by the contribution of the negative correlation between $X^i_{e_3}$ and $X^j_{e_3}$ within the same subblock. Unfortunately, we have $\rho(X^i_{e_3}, X^j_{e_3}) = -3.81\text{x}10^{-6}$ (14), implying that the effect of this correlation is negligible. So, from the variance reduction point of view, there is no gain in taking values of L greater than 753.

Let us consider now the execution time aspect. If we take size blocks growing from 753 to the dagger value $L_d = 251\text{x}3\text{x}2^{18} = 197394432$, the only difference lies in the number of calls to the random number generator. Consider a value L such that $753 \leq L \leq L_d$. Denote by $g(L)$ the number of calls made to the random number generator with block size L, for 753 blocks, that is, for $753\text{x}L$ realizations. We have

$$g(L) = 753\left(2\left\lceil\frac{L}{3}\right\rceil + 23\left\lceil\frac{L}{251}\right\rceil + \left\lceil\frac{L}{2^{18}}\right\rceil\right).$$

For the same number of realizations, the version using a block size of 753 requires $L\text{x}(3\text{x}251 + 23\text{x}3 + 1) = 823\text{x}L$ calls to the generator. The ratio $(823\text{x}L)/g(L)$ has it's maximum for $L = L_d$. Since

$$g(L_d) = 753\text{x}\left(2\text{x}251\text{x}2^{18} + 23\text{x}3\text{x}2^{18} + 251\text{x}3\right) = 112712787681,$$

we obtain at best the ratio

$$\frac{823\text{x}197394432}{112712787681} = 1.441$$

and consequently the effect on the global mean execution time is not significant. For instance, the mean execution time per realization for $L = 753$ computed from the last row in Table 3 is about $1.12\text{x}10^{-4}$ seconds. The executing CPU time of one block with value L_d by the general antithetic algorithm is $2.164\text{x}10^{+4}$ seconds, giving a mean execution time per realization of about $1.10\text{x}10^{-4}$ seconds. So, there is no gain in time neither.

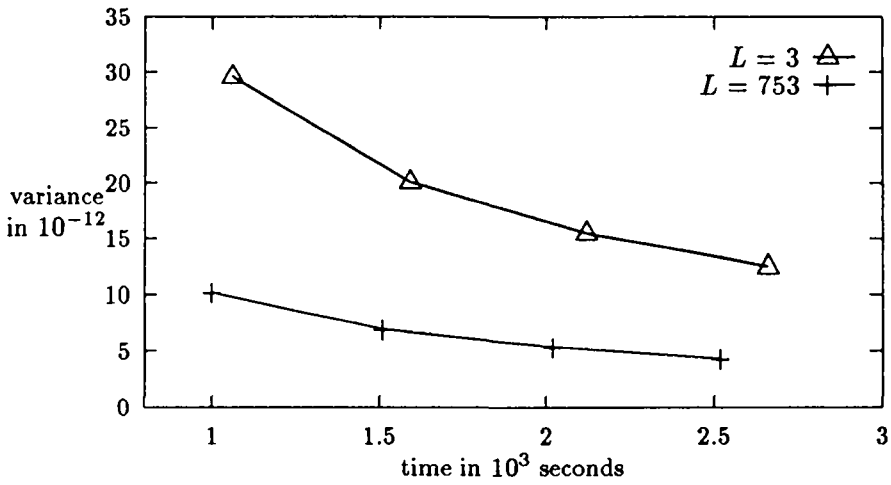

Figure 4: Variance–time curves for the choices $L = 3$ and $L = 753$, on the given version of the Arpanet network.

Moreover, for a fixed execution CPU time (or a fixed number of realizations N) it is of interest to have a sufficient number of blocks in order to use a limit central theorem to derive a confidence interval for the corresponding estimate. If the executing time of one block with size L is too long, a large value of L becomes a constraint with respect to the fixed execution time and the needed numbers of blocks. For instance, with the dagger choice L_d, just one block requires about 6 hours of CPU time ($2.164 \times 10^{+4}$ seconds) and we will need more than one block to estimate the associated variance (formula (5)). With $L = 753$, we give in Table 3 several satisfactory estimates of R and of the associated variance estimator in moderate times.

Resuming, a good choice of L must take into account both the variance reduction and the (mean) execution time aspects. From the preceeding discussions, it follows that L should be a multiple of the s_e's parameters for the edges e having q_e not very small since these edges' contributions are the most important (see the plot of the coefficient of correlation between two states of edge e, formula (14), in Figure 5.) It can be noticed that a second consequence of this choice is that such a value of L is necessarily not very large: For instance, if we take into account just the edges having $q_e \geq 0.1$, then L is bounded by the least common multiple of $\{7, 8, 9, 10\}$ which is equal to 2520. It is possible that for all edge e the parameter q_e is "small" (for instance, if all the components are highly reliable.) In this case, only the time should guide the choice of the block size.

The (mean) execution time reduction needs, when possible, to consider high values of L, to reduce the number of calls to the random number generator. As we have seen, the constraint to respect here is that we need also a minimum number of blocks, depending on the desired precision on the results and on data. Thus, a tradeoff between the two conditions must be found.

Of course, a particularly good situation (from the modelling point of vue) is when L_d is small so that the dagger version can be used. Observe however that from

$$\left\lfloor \frac{1}{q_e} \right\rfloor \leq L_d$$

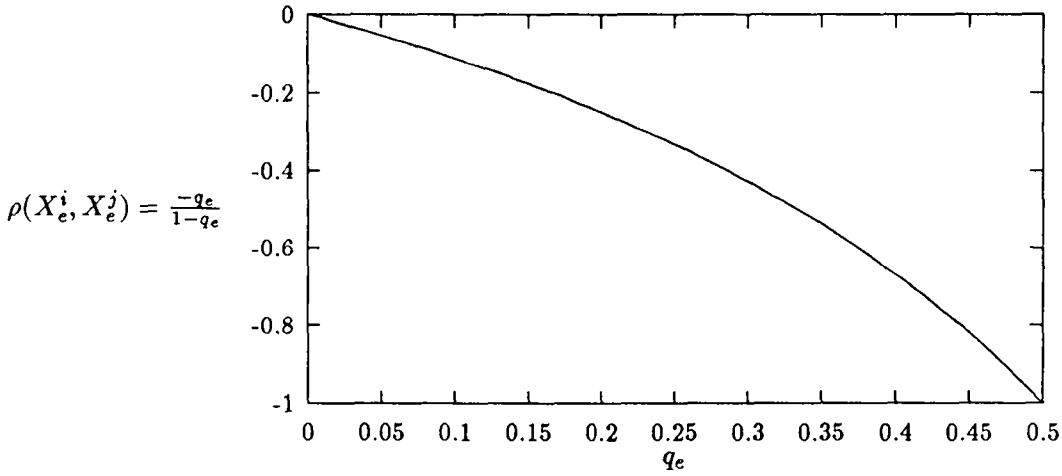

Figure 5: Correlation coefficient between two states of edge e as a function of q_e.

we have

$$q_e \geq \frac{1}{L_d + 1}$$

so, very small values of L_d will occur in the case of "bad" networks having not very reliable components.

A last observation concerning the example: the situation illustrated in it is not unusual. Even if the model is quite regular having, for instance, identical components (same elementary reliabilities), it is of interest to perform some polynomial simplifications as series-parallel reductions or polygon-to-chain reductions (see [14], [3]) as a preprocessing in order to diminish the execution time of the reliability evaluation. These reductions may lead to models where, as stated before, the dagger value of L is very large or even not storable as a program variable.

7 Conclusions

The Monte Carlo evaluation of network reliability measures appears to be very useful in the analysis of communication systems since the exact computation of these measures is extremely time consuming. Even relatively small networks (say, with less than one hundred lines) are often impossible to evaluate exactly.

In this paper, we propose a Monte Carlo technique to evaluate usual network reliability measures. It consists of a straightforward generalization of the antithetic variates method adapted to this specific context (Section 3) and we present several ways to improve its performances (Section 4.)

The algorithm makes no essential assumption on the network and uses only direct data, that is, the topology and the elementary reliabilities. It has an integer input parameter L, the "block size" (Subsection 2.2). The variance of the used estimator and the computing time are both dependent on this parameter. We have shown that the so called dagger sampling plan is a particular case corresponding to a specific value of L, called the "dagger value of L" in the paper and denoted by L_d. This dagger value presents some drawbacks when it is large and this can occur specially

in the (important) case of highly reliable systems. Moreover, in this case the variance reduction is very small. The method proposed here avoids these problems since any value of L going from 1 (corresponding to the crude Monte Carlo technique) to L_d can be used. A good feature of the algorithm is that its time complexity is sensitive to the reliability of the components of the network, which is not the case of the dagger one. The problem of the choice of L is discussed in Section 6 and we show that high values (when L_d is itself high) are not always convenient.

Annex: proof of Theorem 2.1

Let us denote by $X = (X_1, X_2, \ldots, X_n)$ any vector belonging to $\{0,1\}^n$ and by X^i the vector having $n-1$ components, resulting after "eliminating" the ith coordinate of X. If f is a binary structure function on $\{0,1\}^n$, let us denote by $f_i^c(X^i)$ (respectively by $f_i^d(X^i)$) the function associating with X^i the value $f(X)|_{X_i=1}$ (respectively $f(X)|_{X_i=0}$.) If f is the structure function corresponding to the $\mathcal{K}$-terminal problem for some graph $\mathcal{G}$, f_i^c (respectively f_i^d) is the structure function corresponding to the graph resulting after the contraction of the ith edge of $\mathcal{G}$ (respectively after the deletion of the ith edge of $\mathcal{G}$.) It is immediate to check the well known identity

$$f(X) = X_i f_i^c(X^i) + (1 - X_i) f_i^d(X^i).$$

We have then the following result:

Lemma. *Let us denote by f and g two coherent structure functions on $\{0,1\}^n$. Let us denote by $X = (X_1, X_2, \ldots, X_n)$ and $Y = (Y_1, Y_2, \ldots, Y_n)$ two random binary vectors such that*

(i) the random variables $X_1, X_2, \ldots, X_n$ are independent,
the random variables $Y_1, Y_2, \ldots, Y_n$ are independent,

(ii) for any i, either X_i, Y_i are independent, or $Cov(X_i, Y_i) < 0$.

We then have:

$$Cov(f(X), g(Y)) \leq 0$$

and if there exists some j so that $Cov(X_j, Y_j) < 0$, then $Cov(f(X), g(Y)) < 0$.

Proof. If for all i the random variables X_i, Y_i are independent, then $f(X)$ and $g(Y)$ are independent and $\mathrm{Cov}(f(X), g(Y)) = 0$. Assume that for all $i \neq j$, X_i, Y_i are independent, and that $\mathrm{Cov}(X_j, Y_j) < 0$. We have

$$\begin{aligned}
\mathrm{E}[f(X)g(Y)] &= \mathrm{E}\left[\left(X_j f_j^c(X^j) + (1 - X_j) f_j^d(X^j)\right)\left(Y_j g_j^c(Y^j) + (1 - Y_j) g_j^d(Y^j)\right)\right] \\
&= \mathrm{E}[X_j Y_j]\mathrm{E}[f_j^c(X^j)]\mathrm{E}[g_j^c(Y^j)] + \mathrm{E}[X_j(1 - Y_j)]\mathrm{E}[f_j^c(X^j)]\mathrm{E}[g_j^d(Y^j)] \\
&\quad + \mathrm{E}[(1 - X_j)Y_j]\mathrm{E}[f_j^d(X^j)]\mathrm{E}[g_j^c(Y^j)] + \mathrm{E}[(1 - X_j)(1 - Y_j)]\mathrm{E}[f_j^d(X^j)]\mathrm{E}[g_j^d(Y^j)]
\end{aligned}$$

since $f_j^c(X^j)$ is independent of $g_j^c(Y^j)$ and of $g_j^d(Y^j)$, and the same for the pairs $(f_j^d(X^j), g_j^c(Y^j))$ and $(f_j^d(X^j), g_j^d(Y^j))$. In the same way, we write

$$\begin{aligned}
\mathrm{E}[f(X)]\mathrm{E}[g(Y)] &= \mathrm{E}\left[X_j f_j^c(X^j) + (1 - X_j) f_j^d(X^j)\right] \mathrm{E}\left[Y_j g_j^c(Y^j) + (1 - Y_j) g_j^d(Y^j)\right] \\
&= \mathrm{E}[X_j]\mathrm{E}[Y_j]\mathrm{E}[f_j^c(X^j)]\mathrm{E}[g_j^c(Y^j)] + \mathrm{E}[X_j]\mathrm{E}[1 - Y_j]\mathrm{E}[f_j^c(X^j)]\mathrm{E}[g_j^d(Y^j)] \\
&\quad + \mathrm{E}[1 - X_j]\mathrm{E}[Y_j]\mathrm{E}[f_j^d(X^j)]\mathrm{E}[g_j^c(Y^j)] + \mathrm{E}[1 - X_j]\mathrm{E}[1 - Y_j]\mathrm{E}[f_j^d(X^j)]\mathrm{E}[g_j^d(Y^j)].
\end{aligned}$$

Let us take the following notation:

$$P_j^c = \mathrm{E}[f_j^c(X^j)],\ P_j^d = \mathrm{E}[f_j^d(X^j)],\ Q_j^c = \mathrm{E}[g_j^c(Y^j)],\ Q_j^d = \mathrm{E}[g_j^d(Y^j)] \text{ and } \gamma_j = \mathrm{Cov}(X_j, Y_j).$$

After cheking that

$$\mathrm{Cov}(1-X_j, 1-Y_j) = -\gamma_j \text{ and } \mathrm{Cov}(X_j, 1-Y_j) = \mathrm{Cov}(1-X_j, Y_j) = \gamma_j,$$

we obtain

$$\begin{aligned}
\mathrm{Cov}(f(X), g(Y)) &= \mathrm{E}[f(X)g(Y)] - \mathrm{E}[f(X)]\mathrm{E}[g(Y)] \\
&= \gamma_j P_j^c Q_j^c - \gamma_j P_j^c Q_j^d - \gamma_j P_j^d Q_j^c + \gamma_j P_j^d Q_j^d \\
&= \gamma_j \left(P_j^c - P_j^d\right)\left(Q_j^c - Q_j^d\right).
\end{aligned}$$

Since f and g are coherent, we have $P_j^c > P_j^d$ and $Q_j^c > Q_j^d$. This, together with the assumption $\gamma_j < 0$ leads to $\mathrm{Cov}(f(X), g(Y)) < 0$. The last step in the proof is by induction on the number of coordinates j for which $\mathrm{Cov}(X_j, Y_j) < 0$. Let us denote by S this set of coordinates, that is,

$$S = \{j | \gamma_j < 0\}.$$

We have seen that the property is true for $\mathrm{card}(S) = 0$ and $\mathrm{card}(S) = 1$. Assume it holds for $\mathrm{card}(S) \leq k$ $(1 \leq k < n)$ and consider the case of $\mathrm{card}(S) = k+1$. Let us choose any $j \in S$. We have

$$\begin{aligned}
\mathrm{E}[f(X)g(Y)] &= \mathrm{E}\left[\left(X_j f_j^c(X^j) + (1-X_j) f_j^d(X^j)\right)\left(Y_j g_j^c(Y^j) + (1-Y_j) g_j^d(Y^j)\right)\right] \\
&= \mathrm{E}[X_j Y_j]\mathrm{E}[f_j^c(X^j) g_j^c(Y^j)] + \mathrm{E}[X_j(1-Y_j)]\mathrm{E}[f_j^c(X^j) g_j^d(Y^j)] \\
&+ \mathrm{E}[(1-X_j)Y_j]\mathrm{E}[f_j^d(X^j) g_j^c(Y^j)] + \mathrm{E}[(1-X_j)(1-Y_j)]\mathrm{E}[f_j^d(X^j) g_j^d(Y^j)] \\
&< \mathrm{E}[X_j Y_j]\mathrm{E}[f_j^c(X^j)]\mathrm{E}[g_j^c(Y^j)] + \mathrm{E}[X_j(1-Y_j)]\mathrm{E}[f_j^c(X^j)]\mathrm{E}[g_j^d(Y^j)] \\
&+ \mathrm{E}[(1-X_j)Y_j]\mathrm{E}[f_j^d(X^j)]\mathrm{E}[g_j^c(Y^j)] + \mathrm{E}[(1-X_j)(1-Y_j)]\mathrm{E}[f_j^d(X^j)]\mathrm{E}[g_j^d(Y^j)]
\end{aligned}$$

since the four pairs (f_j^c, g_j^c), (f_j^c, g_j^d), (f_j^d, g_j^c) and (f_j^d, g_j^d) have a S–set of cardinality k. This leads to

$$\begin{aligned}
\mathrm{Cov}(f(X), g(Y)) &= \mathrm{E}[f(X)g(Y)] - \mathrm{E}[f(X)]\mathrm{E}[g(Y)] \\
&< \gamma_j P_j^c Q_j^c - \gamma_j P_j^c Q_j^d - \gamma_j P_j^d Q_j^c + \gamma_j P_j^d Q_j^d \\
&= \gamma_j \left(P_j^c - P_j^d\right)\left(Q_j^c - Q_j^d\right)
\end{aligned}$$

and this concludes the proof as in the case $k = 1$. □

References

[1] B.E.Barlow and F.Proschan. *Statistical Theory of Reliability and Life Testing.* Holt, Rinehart & Winston, New York, 1975.

[2] M.O.Ball. Computational complexity of network reliability analysis: an overview. *IEEE Trans. Reliab.*, R-35(3), 1986.

[3] M.O.Locks and A.Satyarayana editors. Network reliability – the state of the art. *IEEE Trans. Reliab.*, R-35(3), 1986.

[4] J.M.Hammersley and D.C.Handscomb. *Monte Carlo Methods.* Halsted Press, Wiley & Sons. Inc., New York, 1979.

[5] R.R.Rubinstein. *Simulation and the Monte Carlo Method.* John Wiley & Sons, 1981.

[6] R.M.Karp and M.Luby. Monte-carlo algorithms for the planar multiterminal network reliability problem. *Journal of Complexity*, 1:45–64, 1985.

[7] H.Kumamoto, K.Tanaka, K.Inoue, and E.J.Henley. Dagger-sampling monte carlo for system unavailability evaluation. *IEEE Trans. Reliab.*, R-29(2), June 1980.

[8] G.S.Fishman. A comparison of four monte-carlo methods for estimating the probability of s-t conectedness. *IEEE Trans. Reliab.*, R-35(2), 1986.

[9] H.Kumamoto, K.Tanaka, and K.Inoue. Efficient evaluation of system reliability by monte carlo method. *IEEE Trans. on Reliab.*, R-26(5), Dec. 1977.

[10] M.El Khadiri, R.Marie, and G.Rubino. Parallel estimation of 2-terminal network reliability by a crude monte carlo technique. In Mehmet Baray and Bülent Özgüç, editors, *ISCIS VI, Sixth International Symposium on Computer and Information Sciences*, Elsevier, Antalya, Turquie, Oct. 1991.

[11] C.J.Colbourn. *The Combinatorics of Network Reliability.* Oxford University Press, New York, 1987.

[12] F.Harary. *Graph Theory.* Addison-Wesley, 1969.

[13] A.Satyarayana and M.K.Chang. Network reliability and the factoring theorem. *Networks*, 13, 1983.

[14] A.Satyanarayana and R.K.Wood. A linear-time algorithm for computing k-terminal in series-parallel networks. *SIAM J. Comput.*, 14(4), Nov. 1985.

LISTE DES PUBLICATIONS INTERNES IRISA 1992

PI 624 SIGNAL AS A MODEL FOR REAL-TIME AND HYBRID SYSTEMS
Albert BENVENISTE, Michel LE BORGNE, Paul LE GUERNIC
Janvier 1992, 22 pages.

PI 625 ON THE CENTRAL-LIMIT THEOREM FOR TRACKING ESTIMATORS WITH SMALL GAIN - INFINITE HORIZON CASE
Bernard DELYON, Anatoli JUDITSKY
Janvier 1992, 16 pages.

PI 626 A MONTE CARLO METHOD BASED ON ANTITHETIC VARIATES FOR NETWORK RELIABILITY COMPUTATIONS
Mohamed EL KHADIRI, Gerardo RUBINO
Janvier 1992, 28 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399